MicroWorld Z80 Editor / Assembler

microbee users manual

Australia's Own Personal and Educational Computer
Developed to provide the fun way into microcomputers.

Cat. No. 250. 006

MicroWorld Z80 Editor/Assembler

Instruction manual

for MicroBee DGOS and CP/M versions

CONTENTS/INDEX

INTRODUCTION

This package consists of a "line oriented" text editor with automatic line numbering, an assembler which will generate Z80 machine code from standard Zilog Mnemonics as specified in the "Zilog Z80-Assembly Language Programming Manual". Macro expansion however, and certain arithmetic operators listed in the Zilog Manual are not supported.

The Editor files may be saved to, or loaded from, cassette using the Save and Get commands (these functions operate to and from disk in the CP/M version).

Multiple files may be present in memory at the same time. Lines from the primary file may be copied and appended to the specified secondary file, or the entire contents of a specified secondary file may be merged back into the primary file with automatic renumbering of the primary file if required.

Specific lines in the currently "open" file may be accessed by their line number or by cursor control. Once accessed the lines may be altered, appended to, replaced, or deleted. Lines may be inserted into the file by simply assigning a line number appropriate to the location you wish it to take up in the file. Likewise a line, or block of lines, may be manipulated or killed by specifying the block of lines involved.

Global search and replace functions are supported by the editor, each occurrence in the run is reported as it is found.

Up to 14 lines on either side of the "current line" may be inspected, without moving the line pointer, by use of the "View" feature.

Files may be typed to a printer with the line numbers stripped off, allowing the Editor to be used for letter writing.

The Assembler produces its object code directly into memory ready for immediate execution. An offset feature allows the object to be located in a location other than where it would normally reside, to prevent overlaying important memory areas during assembly. Since a three pass technique is used the object code may even overlay the source file if desired.

Assembler labels may be up to 6 characters in length and provided that they start with an ascii character may have almost any other character imbedded within them. Source listings generated during assembly may be suppressed or directed to either VDU or printer.

Full error reporting is provided even when source listing is suppressed, a "wait on error" function is allowed, with conditional return to the Editor automatically set up on the line containing the error.

Print directives are provided to allow the listing to printer to be turned on or off under software control, this allows the printing of partial source listings. Listing of the object code for long strings or data statements is automatically suppressed to conserve paper, a "switch" is provided to allow listing in full if desired.

WHAT IS AN "EDITOR" ?

If the preceding introduction was full of strange new terms, read on, if not you may skip the next couple of chapters. As you read this manual you will notice that whenever a new term is introduced into the text it is shown in CAPITAL LETTERS, if after reading the paragraph you are still not sure of the meaning of the word try looking it up in the glossary of terms at the end of the manual.

Essentially an EDITOR is a program which allows you to create in memory a "file" containing text. You may use the editor to enter, correct, or rearrange your file and to save and retrieve it from some storage medium, usually (disk or cassette). There are two types of editors in common usage, these are known as "SCREEN" editors and "LINE" editors. The screen editor is the more sophisticated of the two, it uses the T.V. screen (VDU) as a "window" into the file. To visualise this consider a piece of card with a cutout that is the width of the printing on this page and about 2Ø lines high, you can slide the card up and down the page and see the window effect in action. In the screen editor you have a flashing point of light (cursor) that can be moved anywhere on the display. If you try to move the cursor above or below the screen, the window is automatically moved up or down the file so that the cursor is always on the screen. As you insert or delete letters or words, the entire screen display is redrawn to show how that part of the file is layed out. The really sophisticated screen editors provide features like justifying the left and right edges of the lines so that they are both straight as in this manual. These editors are usually referred to as "word processors", they are very expensive programs to purchase, use up a large amount of processor memory, and require a "DISK" system for file storage. This is because the editor itself is so large that there is not much room left in memory for the file, therefore it must be edited directly from the disk. Enough of this dreaming of what might be, you are stuck with a "line" editor.

The line editor is quite different in concept, it considers your file to be a collection of LINES, each line has a maximum length, usually the width of your TV screen; these lines are considered to be in sequential order usually with a LINE NUMBER attached to them. The line numbers are usually arranged to increment by 1Ø as each line is inserted into the file. This is done to allow extra lines to be inserted between existing ones. The main disadvantage here is that if you need to insert more than 9 lines between any pair of existing lines you must renumber the file and then any printout you may be using to edit your file from will disagree in numbering. In practice this is not a major problem.
To use the editor to create a file, you just type the required lines whilst in INSERT mode, remembering to press the return key at the end of each line. Up to this point it is nearly as easy to use as a screen editor, however here the similarity ends. If you wish to change something in the file, you may direct the editor to go to a line number that is near where you wish to make the change, move the cursor (now called a CURRENT LINE POINTER) up or down the file till it points to the line you wish to change, and give an appropriate command to DELETE, EDIT, INSERT, REPLACE, etc. If your command was Edit, this version of the Editor is provided with a 'SUBEDIT' package which gives you some of the advantages of a screen editor, however you are confined within the bounds of the line you are currently editing.

To make life a little bit easier (who said it wasn't meant to be), we have provided commands to allow you to search your file for particular words or phrases, and if required replace them with alternate ones.
Although line editors usually have line numbers associated with them, the editor does all the allocation and distribution of them, Since over 65000 are allowed you will always run out of memory long before you run out of numbers, unless you number your file in increments of a hundred or so. The file may also be typed with the line numbers automatically suppressed for letter writing etc.

WHAT IS AN "ASSEMBLER" ?

If you are already familiar with assemblers, here is another chapter you may skip, if not you should persevere a bit longer and wade through this chapter also.

Although a computer may appear to be an extremely complex and intelligent piece of equipment, it is in reality a number of PERIPHERALS (VDU, keyboard, memory, printer, cassettes, etc) all connected to the CPU chip (Central Processor Unit). This tiny silicon wafer in a 40pin I.C. package has one claim to fame; it can do a small number (a couple of hundred) simple tasks, very quickly, and very reliably. It is instructed to do each of these tasks by a sequence of numbers which when strung together as a series of tasks produce a PROGRAM. It is the program, not the computer which produces the illusion of intelligence. Whilst the CPU unit in your MicroBee knows that the sequence C30080 means to go to memory address 8000hex and start "RUNNING" the BASIC interpreter we have installed there, you as the user cannot be expected to know what all the hundreds of combinations of numbers will mean.
It will be obvious by now that if you wish your computer to do any useful work for you it must be given programs to do do these tasks, so how do you provide these? You could go out and purchase all your programs. That is assuming they are available, and are within your financial resources. An alternate approach is to purchase a "HIGH LEVEL LANGUAGE" such as PILOT, BASIC, PASCAL, FORTRAN, COBOL, etc. These programs allow you to list your program requirements using a group of "english" like words sometimes referred to as MNEMONICS. When run, these programs "interpret" your words and provide the functions required. The main disadvantages with this approach are the cost of the interpreters, the amount of memory they require to operate, restrictions in the tasks that can be performed, and the speed of execution is fairly slow. The speed factor is particularly important if you wish to play real time games such as "SPACE INVADERS". Whilst COMPILER versions of most of the above programs are available they are not suitable for small systems.

Having reached the subject of compilers we can now discuss the simplest compiler of them all, the ASSEMBLER. I use the word simplest only in the sense that you can't just say PRINT and have the program go away and produce a complete SUB PROGRAM to do a 'print message' function as you would with an interpreter; however in this simplicity lives its versatility, you have complete control over each and every instruction in the program.
As with an interpreter you provide a SOURCE FILE consisting of mnemonics, in this case each LINE of the source describes ONE instruction that you wish

the computer to perform, you may also provide LABELS on any line so that you may later instruct the computer to go to this point in the program. Comments may also be inserted on the line so that you or your friends may later be able to work out what was intended when you wrote it. As well as referring to a location within a program, labels may also be assigned absolute values. This has the advantage of allowing you to alter this value everywhere in the program by just altering the value at the location it is declared. Another reason for assigning values to labels is that it is often simpler to remember a name than the value assigned to it, eg it is easier and clearer to say TAB or SPACE than to remember that they are Ø9 and 32 respectively.
In case you haven't already guessed, the assembler interprets your source code and produces, directly in memory, the sequence of numbers referred to as MACHINE CODE or OBJECT CODE that make up the program ready for your processor to execute at full machine speed. It is good practice to save your source file on cassette in case you wish to make alterations in the future, or wish to assemble the program to operate at a different location in memory. The latter can be done by simply changing the origin (ORG) address and re-assembling the file.

SYSTEM REQUIREMENTS

The Editor/Assembler can be provided to operate under four separate system environments.

1. Standard DGOS ver 1.4 or equivalent
   ØØØØ to ØØFF hex Printer patch
   Ø1ØØ to 1AFF hex Editor/Assembler
   1BØØ to 1DFF hex Personality module (handles all I/O functions)
   1EØØ to 1FFF hex Scratch pad RAM area
   2ØØØ upwards Editor source files

2. Standard MicroBee
   ØØØØ to Ø1FF hex Basic scratch (do not overlay)
   Ø2ØØ to Ø3FF hex Editor/Assembler scratch
   Ø4ØØ to ØFFF hex Area for generated object code
   1ØØØ to upper limit of RAM Editor source files
   8ØØØ to ØBFF hex Basic interpreter
   CØØØ to DFFF hex Entire software package plus some monitor functions

3. Standard CP/M version
   As per DGOS version except printer patch is not required since ALL I/O is handled via BIOS, and all files are stored on disk.

4. Non standard version for other Z8Ø systems.
   Since all I/O functions are handled by the personality module you may customise the standard DGOS version to suit your monitor requirements. If your system does not have free RAM in the areas required, a special version can be assembled into any 8k memory location.

SOME "HANDS ON" EXPERIENCE

By far the best way to come to grips with any new processor operating system is to have someone demonstrate the program in actual use, and then try it out by yourself under supervision. Unfortunately we can't quite do this here but we can guide you through one of the examples in the appendix to this manual. Hopefully this will give you a better understanding of the commands and their use when we describe them in detail in the next chapter.

First you must get into the program. If you have a MicroBee just call the editor from Basic, see your Basic manual for the exact command to use. If you are using a CP/M version call it off disk with the command EDASEM. If you have a DGOS version load the tape and G 100 from the DGOS monitor. You should now have a cleared screen with the message "MEMORY SIZE?" displayed, answer 4000<ret> **[Whenever you see <ret> shown in this description it means press the carriage return key]**. The screen should now be cleared again, and the editor will announce itself by the message "EDITOR ASSEMBLER" and wait for a command to be entered [the "*" is the editor PROMPT character indicating it is ready to accept a command].

Type Q<ret> the program will respond 2000 2000 2000 3FFF
If you have a MicroBee the 2000s will be 1000s. The first address indicates that the file will start at 2000hex. The second address is the location of the CURRENT LINE pointer and is now pointing to the start of the file. The third address indicates that the file ends at 2000hex (which is logical cause there ain't no file there yet) the last address is one byte lower than the upper limit of memory that you set on entry, the editor will not attempt to use memory above this point.

Now to enter our sample file:- After reading this paragraph, type I<ret>
The program responds 00100 ▮ (▮ represents the cursor)
You may then type in the file called "TEST" in appendix A. The program as listed contains some deliberate errors to demonstrate editing and error handling. If you make a couple more errors typing it in don't worry, we will show you how to correct them at the end. Don't forget to press <ret> at the end of each line. You will notice that each line is tabulated into columns, this tabulation is normally produced by pressing the TAB key (or Control and I simultaneously) at each break instead of the SPACE key. This neat layout format is to make the code more easily read by you and is not essential for the assembler, it is quite happy with just single spaces between each FIELD. If you prefer the neat layout but do not have a TAB key, and find typing Cntrl&I awkward, you may use multiple spaces to do the tabulation, bear in mind though that all these extra spaces will greatly increase the size of your source file. If you make a complete bollox of it first time just reset the processor and start again. Notice that the editor inserted the line numbers for you. N.B. WHEN YOU HAVE FINISHED THE LAST LINE TYPE ^A or ^C. THIS MEANS PRESS THE CONTROL KEY AND THE A or C KEYS SIMULTANEOUSLY, this will get you out of the INSERT mode and back to the editor.

You should now be back in the editor with the "*" prompt showing and the test file typed in, with or without a few more typing errors of your own. It is suggested that you save the file on cassette now to save having to retype

it all back later. Do this by starting the recorder and typing S "TEST"<ret> The editor will reprompt with a * when it has completed the save.

To examine the file you have typed in, type P#:*<ret> and the entire program should be listed to the VDU (a bit too quickly to read). Now type P#<ret> and only the top line will be printed, each time you press the Line Feed key one more line will be displayed, by this means you can step through your file line by line. If your keyboard has a '^' key (called circumflex) press it and you will see that you will step backwards through the file one line each time you press the key. Back to the top of the file again by P#<ret>, now press P<ret> and you can step through the file a screen full at a time.

Let's assume you made a mistake in line 200, the easiest way is to REPLACE it by typing R200<ret> and type the line in again, end the line with a return as normal. Notice that you press return twice, first after the command and again at the end of the line. The program should return to the editor with the message "no room between lines". Normally the replace mode would allow you to replace the line with more than one line, but here after replacing line 200 the counter was incremented to 210 and the program would have destroyed an existing line if you had continued. If you wish to put in more than 1 line use the command R200,2<ret> this would allow you up to 5 lines numbered 200, 202, 204, 206, 208 before it aborted out, you can of course finish earlier by the pressing Control and C (or A).

There are two important points to note, here firstly you must not press the Control C at the end of the last line before you exit or it will not be inserted into the file, end the line with a <ret> and type the ^C when it prompts with the next line number (which you do not wish to enter). Secondly if you used the format R200,2 your line number increment counter is now set up for steps of 2, and all future changes will be in steps of 2 until it is reset to some other value. The step size may be altered by appending a comma and new step value to the REPLACE, INSERT, or RENUMBER commands. You will have noticed that the editor starts up with the first line number as 100 and steps of 10. These are the DEFAULT values.

If you made any typing errors you may either fix them now with the replace command, or wait till after we have examined the EDIT function and edit out your errors.

Lets EDIT line 100, type E100<ret> the editor prints 00100 with the cursor after the number, ready for a sub edit command. Press the space bar a few times, you will notice that the characters are gradually revealed on the line, the space bar is doing a "non destructive" forward space function. Now type L and the entire line is displayed and the cursor is redisplayed at the start of the line ready for another edit of the line. Now type X and the line is displayed with the cursor sitting at the end of the line ready for you to append a comment. Now type FRED LIKES NUTS<ret> and edit the line again, you will notice that your comment has been added to the end of the line. Use the X sub command to go to the end of the line again and use the backspace key to step back over your comment then press return, your comment has been removed from the line. This demonstrates that backspace is "destructive". If the error you wish to correct is near the end of a line it is often faster to go to the end of the line, backspace over the error and retype the end of the line.

The line should be back to something like its original condition. Now type E100<ret> again, press the space bar to reveal the ";" as before and type 4DIANOTHER<ret> the 4D instructed the editor to delete the next 4 characters, then you told it to insert the following characters, and the "<ret>" caused it to finish the edit of the line. Your line now reads 00100 ;ANOTHER program for "skywriter". If what you wished to replace was the same length as what you were replacing you could have typed 4CBILL<ret> which would have CHANGED the next 4 characters to BILL. There are many other commands in the sub edit mode but what you know now should allow you to fix up any mistakes you may have made whilst typing in the test file, we will deal with the rest in detail later.
One final hint at this stage, if you really make a muck up of your file and wish to start again from the version saved on cassette type Z<ret> to erase the file in memory, then type G "TEST"<ret> and play the tape you recorded before. After the tape has loaded you should be back at the Editor-Assembler message with the file in memory ready to edit again.

Assuming that all has gone well and your file looks exactly like the one in the appendix we are ready to try to assemble it into a working program. If the version you previously saved to cassette is not correct, it would be advisable to resave the file now.

To do a trial assembly type A/NO/WE<ret>. The 2 letter groups following each slash are termed SWITCHES and these direct the assembler to do specific functions during the assembly. The 2 switches used here are:-
NO means No Object code is to be output into memory.
WE means Wait if you find an Error. (and report it)

If all has gone according to plan you should have seen several screen fulls of SOURCE LISTING scroll up the VDU, and when line 400 was reached, the assembly was stopped and an error reported. If you now type Control and C simultaneously you will find yourself back in the editor with the CURRENT LINE POINTER at the start of the line with the error in. Now type E<ret>L and the line will be displayed. What the assembler was complaining about was an unknown label called STRNG, you will notice that when we defined it in line 480 it was called STRING. To fix this up type X then two backspaces and ING<ret> which should fix this error. Now try the assembly again.

Again the assembler should stop at an error, this time on line 490, now the error is reported as an "argument error". What we are being told is that the value 0A is incorrect, remember that in its present setup the assembler is expecting all values to be decimal and 0A is a hex number. The fault can be fixed by changing the value to 10 (the decimal equivalent of 0Ahex) or by simply adding a H to the end of the line to tell the assembler that we require a hex value. This time the file should assemble without any errors.

If any other errors are reported you should be able to work them out by reference to the chapter on "assembler errors" and rechecking your file against the listing in the appendix.

I cannot stress the concept of backup too strongly, now that you have a working file, save it to cassette.

Note that a list of all the labels, with their values was printed at the end of the assembly, this is called a SYMBOL TABLE. As a final check before we

generate the object program check that VDU is shown as F000 in the symbol table and START is shown as 3000. It is so easy to forget to put the H after hex values and unless the value contains an ascii character the assembler will happily accept it as decimal, and we certainly don't want our program to be assembled at address 3000 decimal as it would damage part of the assembler itself.

The final step now is to assemble the file and generate the program in memory. To do this type A/WE<ret> it is always good practice to use the WE switch since it is better to stop the assembly if an error occurs than to miss seeing the error reported, and try to run a faulty program.

When the assembler has finished and returned to the editor prompt you may attempt to run your program with the eXecute command X3000<ret> the top 4 lines of the screen should now be cleared and a small rocket should slowly circulate round the top of the screen producing a banner with the alphabet on it. In this simple program I have not attempted to allow any exit, it will continue ad nauseam till you reset the processor. Resetting the processor may with some systems destroy the source file, and you may need to reload from cassette to continue experimenting with the source file.

EDITOR INSTRUCTIONS

We will now deal with the editor commands in detail, treating each in alphabetical sequence, rather than in any order of priority. Note that only the Z, ZS, O, OS, Q, B, X, G, and I commands may be used when no file is present in the editor.

B The Bye command leaves the editor and returns to BASIC or system monitor depending on which version you are running.

C This is the change command, the format is C/string1/string2/*<ret> It will start at the line AFTER the current line pointer and change all occurrences of string1 to string2, the strings do not need to be of the same length. If the star is omitted from the end, the change will only be done once. In either case the change will only occur once per line. If you wish to do the change throughout the entire file don't forget to go to the top of the file first. The maximum length of either string is 15 characters. To delete a string use C/string1//*<ret>.

CO The COpy command takes a block of lines in the primary file and appends them to the end of the currently open secondary file. The primary file is not altered by this command and it may be used repeatedly to build up a required secondary file. After each copy is completed the STATUS of the secondary file is displayed as a reminder to the user that the editor does not check or protect the secondary file. You are advised to read the notes on secondary files at the end of the chapter. The format of the command is COxxxx:yyyy<ret>.

D The delete line command has 2 formats, Dxxxx<ret> or Dxxxx:yyyy<ret>. In the first case only the specified line is deleted, in the second case all lines between (and including) the specified line numbers are deleted from the file. As with all commands which allow a line number to be specified, if no line number is given, the current line is assumed. eg D<ret> deletes the current line. We do not recommend the use of WILD CARDS with the delete command, D#:*<ret> is not the best way to delete the entire file, use Z<ret> instead.

E This command enters you into the EDIT mode, see the separate list of commands available in this mode. E100<ret> will edit line number 100 and E<ret> will edit the current line. The edit command only works on one line at a time, you cannot chain through the file in edit mode.

F The find command will search the file for a given string of up to 15 characters, starting at the line AFTER the current line. If found the line is displayed, and becomes the current line. The format is F/string1/<ret>. The string used is remembered and F<ret> will default to the last used string. ie F<ret> may be used repeatedly to search the file for all occurrences of a string. There is one exception to this rule, The C command shares the same string buffer as the F command and after a change command, the default string for the F command becomes the first string of the last change command.

G This command will load a file from cassette (or disk if CP/M version). The command has two formats, G"name"<ret> or G*<ret>. In the first case the tape file must have a title which matches your spec, the file spec may be shorter that the tape name eg G"FR"<ret> will accept any tape name starting with FR. When a tape name is specified the file must have an S FILETYPE. All tapes made by the editor will automatically have this file type encoded into them. If you forget to use the quotes around the name, the program will complain about an incorrect format. If you do not know the name of the file you wish to load, you may use the global form of the command eg G*<ret> This will load ANY file with ANY filetype, for this reason the global form should be used with caution. In either case the tape file will be appended to the end of the currently open primary file. If you are not deliberately trying to build up a multiple file, or concatenate a file from tape, always ensure that the currently open file is empty (use Z command) before loading a tape file.

I The Insert command allows you to add new lines to the file, the format is Innnn,ss<ret>. The comma ss is an optional parameter to allow you to specify the STEP size between line numbers, if not used the current step size will be assumed. The line number, if given, is the location in the file where your next line will be inserted. If this line number already exists in the file, the step value is added to the specified number and will be used if it will fit before the next line number existing in the file. This means that if the line already exists the editor will default to attempt to insert the new line after the existing line. If no parameters are given eg I<ret> the editor will default to inserting after the CURRENT line. Once in insert mode the editor continues to insert lines adding the step size for each, until you request to exit by typing CntrlC or CntrlA, or until on adding the step size to generate the next line number, the number exceeds the next existing line number in the file. In this case the editor aborts out of insert mode with a message "no room between lines".

L As per the P command except the output is directed to the line printer instead of the VDU.

M The merge command will copy the entire contents of the currently open secondary file back into the primary file. As each line is copied back it is assigned a line number with a step value of 2. At the completion of the merge the remainder of the primary file is renumbered with steps of 2 until the file is back in step. This causes minimum dislocation of numbering in the primary file. The format of the command is Mnnnn<ret>. If no line number is given, the program will default so that the merge occurs after the current line.

N This command will renumber the complete file the format is Nnumber,step eg N100,10<ret> will start at the top of the file calling it number 100 and set the remaining lines in incrementing steps of 10.

O This attempts to re-open an existing file at the specified address, if a valid file is found, it is given the status of primary file. If no address for the open is given or if the address is outside the allowable bounds for a file, the default address (1000 or 2000) is used. It is always good practice to query the status of the file with the Q command after it has been opened, since it is possible (but not likely) to find rubbish in memory that satisfies the requirements for a file. This command is normally used to temporarily set up on an old secondary file to allow editing or examination of that file.

OS This attempts to re-assign the open status of an existing secondary file, it is normally used to switch between secondary files when more than one is in memory at one time, or to re-establish a secondary file after a trial assembly (assuming that the assembly didn't damage the secondary file).

P Print specified lines to the VDU. Since full WILD CARDS are allowed in this command, it has many variations in format. The general form is Pxxxx:yyyy<ret> where all lines between the two line numbers are displayed to the VDU. In all instances of the P command the last line printed becomes the current line pointer. If only one line number is given. eg P150<ret> this line is displayed, and becomes the "current line". This is the normal method of moving around in the file. If no line numbers are given, eg P<ret> the next 15 lines are displayed from the current line, this is the normal method of stepping through the file. The wild cards normally used are:
refers to the start of the file.
. refers to the current line.
* refers to the end of the file.
Thus P#<ret> would return to the start of the file. P.<ret> would reprint the current line. and P*<ret> would move to the end of the file. These wild cards may be used as both arguments eg P#:*<ret> prints the entire file from beginning to end.

Q Query the STATUS of the currently open file, 4 addresses are given in the display, START of file, CURRENT LINE POINTER, END of the file, and UPPER LIMIT of memory. Although the upper limit is normally the top address of your system RAM it is sometimes convenient to specify a lower address to have some 'protected' memory in the system.

R Replace the specified line in the file, the step value may also be redefined with this command. After replacing the requested line the editor will automatically go to insert mode if the step size will allow another line before the next existing line in the file. Normal format is Rnnnn,ss<ret>. If no arguments are given, the default is to replace the current line.

S Save the currently open file to cassette, the form is S "NAME"<ret> the quotes and a file name of up to 6 characters are compulsory. An appropriate error message is generated unless the format is correct.

T Type the file to a printer as per command L, except line numbers are automatically stripped off. This is mainly used for letter writing.

V View the 14 lines around the current line without moving the current line pointer. If an argument is given. eg V9<ret> the current line will be the ninth one in the display. If the start or end of the file occurs within the specified view, the program adjusts the view offset accordingly. If no argument is given. eg V<ret> the current line will

be in the centre of the display. This command is very handy during editing sessions to compare the overall edited effect of the area you are working in.

X This is the eXecution command, it may be used to test run your new program or to jump to some other point in memory eg to a monitor etc. The format of the command is Xaddress<ret>. If no address is given it will default to system monitor. In the case of "MicroBee" this is a MACHINE level monitor provided with this package.

Z This command creates a 'new' file, if an address is given eg Z2000<ret> the file will be created at the specified (hex) address. If the address is outside the allowable file area, or if no address is given the file will be opened at the DEFAULT address. The file when open is given the status of primary file.

ZS This creates a SECONDARY file at the specified address eg ZS3000<ret>. If no address is specified the file will default to 1k above the current end of the primary file. Note that the editor does not do any checks to ensure that it does not destroy the secondary file if your primary file expands upwards in memory, you must ensure that sufficient room exists. You may have any number of secondary files in memory at any time, however only the currently 'open' secondary file will be used for copies or merges. No secondary file may be opened at an address lower in memory than the currently open primary file. After assemblies all secondary files are considered 'closed'. They may, or may not have been damaged by the assembly process, this depends on the lower limit of SYMBOL TABLE expansion, and where any object code may have been produced. Since secondary files are 'temporary' this restraint should not cause trouble.

The circumflex causes the current line pointer to be moved backwards (towards the start of the file) by one line. The preceeding line is printed and becomes the current line.

The reverse apostrophe (code 60h) is the means of 'freezing' the display during scrolling, the scroll may be resumed by pressing any other key.

L/F The Line Feed key causes the current line pointer to be moved forward by one line. The next line in the file is printed and becomes the current line.

CntrlA This combination of two keys pressed simultaneously (actually the control key is pressed first and held down whilst the A key is pressed) allows exit from the INSERT mode, and the SUB insert string mode. operation is resumed at the editor command level.

CntrlC Same as Cntrl A, included for CP/M compatability.

TAB The TAB character when inserted into a file causes the VDU to print the following word at the next location which is a multiple of 8 characters from the left edge of the screen. This is used to tabulate the file for easier reading. in all cases the program will allow one or more spaces to be used in place of a tab. If your keyboard does not have a TAB key you may simulate it by using Control and I keys simultaneously.

B/S The backspace key moves back one character in the line, it is "destructive" and any characters back spaced over will not be included in the file line.

If you wish to insert a comment on a line, the comment must be preceeded by a semicolon ";". All characters to the right of a semicolon are ignored by the assembler. If you wish to use blank lines to break up a source listing for easier reading, you may use either a single semicolon on the line, or

insert a NULL line (<ret> only). In either instance the line is ignored by the assembler.

SECONDARY FILES.
One of the problems in using a line oriented editor is encountered when you wish to move blocks of lines from one part of a file into another, or if you wish to create a new file from sections of one or more existing files. To make this task easier we have introduced the concept of secondary files into the editor. At first it was thought that we would arrange for the editor to keep track of the start and finish pointers for both files, this caused two problems. It reduced the amount of space available for symbol table expansion and we soon realised that it was convenient at times to have more than one secondary file. Like many good ideas the whole thing got out of hand. The editor started to grow alarmingly, and run time was being slowed down due to the amount of checking that had to be carried out all the time. So like it or not there is **NO** protection on the secondary files. These files must be considered as temporary only.
To use a secondary file you simply define an address above the end of your main file and declare it available for use ("open") with the ZS instruction. To transfer material to this secondary file you use the COpy instruction. The editor verifies that a secondary file has been declared open, finds the end of this file and simply copies the lines from the primary file to this location and appends an end of file marker to the tail. In this manner you may just keep appending lines from the main file to the end of this imaginary file. The astute readers will have realised by now that a major problem is occurring, the line numbers are probably all scrambled up and not necessarily in ascending order, and duplicates of some line numbers may exist, so any attempt to print or edit this file would be disastrous. For this reason, whenever you setup on a secondary file using the O command, you **MUST** do a renumber of the file before executing any other command. This file will now be considered to be the primary file and you may now edit, insert or save the file as desired. You will not damage the OLD primary file as it is lower in memory space. Only one restriction applies, you cannot specify the old primary file to now be a secondary file for the purpose of further copies or merges. You may however return to the old primary file with the O command, and redefine this file to secondary status with the OS command.

SUB EDIT INSTRUCTIONS

Whilst in EDIT mode, the normal editor command set is not available, and a separate set of commands are used, it must be stressed that these commands are not DELIMITED by a <ret>. In edit mode the return key has a special function, it ends the edit, inserts any changes made, and returns to normal editor level.

A Ignore any changes already made, restart the edit of this line using original line from file.

nC Change the next 'n' characters in the line from what they are at present to whatever you type next. You must now type the specified number of characters, no more, and no less. eg if you are at the start of the word ELEPHANT and you enter 4CFRED the word will read FREDHANT. If you had typed 4CFREDD the word would have changed to FREDANT since the last D would have been interpreted as a delete 1 character command.

nD Delete the next 'n' characters from the line, no ARGUMENT is required after the command, it is simply to delete characters. If no number of characters is given the default is one.

E End edit, insert changes into the file. It is normal to use the return key to provide this function. The E sub command was only provided for compatibility with other editors.

H Delete all characters after this point on the line and enter the I sub command to allow extra characters to be inserted. If you only wish to kill the rest of the line, ie remove a comment, use the format H<ret>

I Insert all further characters typed into the line. You must use the return key when you have finished typing the new text, which will also end the edit. if further changes are required, you must re edit the line

nKx This command will kill (delete) all characters from the current character up to the n'th occurrence of the character specified in position x. eg 5Ky would delete all characters on the line till it reached the fifth y on the line.

L This command lists the entire line to the VDU and returns the edit pointer to the start of the line. It may be used to inspect the line before the edit begins, or to view the effect of the edit to date. The edit is not terminated, nor is the line in the file updated, it simply allows you to inspect the line as it exists in the edit BUFFER.

Q Quit the edit, return to the editor, do not alter the original line in the file.

nSx This command allows you to quickly move the pointer to a specified position on the line. eg 5Sa will move the pointer to the fifth 'a' on the line.

X This is the APPEND command. The pointer is moved to the end of the line and insert sub command is entered. The command is normally used to insert comments onto the end of a line.

SPACE This key will move the pointer one character to the right, it is non destructive and is used to step forward along the line. There is no non destructive back space key to move left. If you wish to go backwards along a line, use the L command to list the line and recommence from the start of the line.

B/S The backspace key is destructive, any characters backed over will be deleted from the line.

<ret>End the edit and replace the line in the file with the one we have just edited. Control is returned to normal editor level with the current line pointer still set up to the line we have just edited.

ASSEMBLER INSTRUCTIONS

The Assembler is a three PASS device, this means that your source file is read from beginning to end three times by the assembler during the assembly process. On the first pass the LABELS are recorded in a special list, called a SYMBOL TABLE, and addresses or values are assigned to them. This list starts at the TOP OF MEMORY address, given on entry to the editor, and grows down through memory as each new label and value are added. Checks are made to ensure that the symbol table does not "crash" into the end of your source file. If this is about to occur, an error message "Symbol table OVF"

is generated, and the assembly is aborted. At the end of pass 1 the assembler knows the location and value of every SYMBOLIC REFERENCE in your source file. Pass two is used to interpret the MNEMONICS and assign the values to all symbolic references in the argument field. It is during this pass that most errors will be detected, the source listings and printouts are also generated at this time. The third pass is used ,if required, to generate the OBJECT program into memory.

To commence assembly you issue an A command from the editor. The format of this command is fairly exacting as an OFFSET may be specified, and a number of SWITCHES may be included to direct the assembler to perform specific tasks during assembly.
The offset allows the assembler to locate the output program code at a different address in memory to the address that it is intended to operate at. This feature will not normally be required by Microbee users since a special area at 400hex has been allocated for them to generate and run their programs in. DGOS and CP/M users however, will generally wish their object program to go at 100hex. If they attempted to assemble directly to this location the Editor/Assembler would be partially destroyed by the code as it was output, with catastrophic results. They will therefore specify an offset that places the code in a safe location. The offset value is simply added to the address that each byte will be stored at. eg an offset of 1000 specified for a source ORGed at location 100hex will cause the output code to be stored starting at location 1100hex. Reverse offsets are possible due to address wrap around at FFFFhex, this means that an offset of 0F000 will cause the code to be stored 1000hex bytes lower in memory. Note that the offset address is always in hex, no H is required after the address, and any value commencing with an alpha character must be preceeded by a zero. After assembly, any code generated with an offset must be moved to its correct location before it may be run.
There are up to six SWITCHES that may be specified in the assembly command line. Each switch when used is identified by typing a slash before it. The switches are:-

WE This switch directs the assembler to stop whenever an error is detected during pass 2 of the assembly. The error is displayed to the VDU, and the assembler waits for a direction from the keyboard. If Control C is pressed, the assembly is aborted, and command is passed back to the Editor with the error line set up as the current line so that you may examine or edit the line. If C is pressed (not control C) the WE switch will be cleared and assembly continues, however the assembler will not stop on further errors. If any other key is pressed the assembly continues with the WE switch still active.

NO This directs the assembler NOT to output any object code during the assembly. The NO switch should always be used for trial assemblies until you are sure that no errors exist.

NL This suppresses listing to the VDU, errors if encountered will however still be displayed. The listing may alternatively be turned on and off by special print directives in the program. See "Pseudo Mnemonics".

NS Do not list or print the symbol table at the end of the assembly.

LP Direct all listings to the line printer device instead of to the VDU. In the DGOS version, if no printer is available the output is to a "slow" VDU driver. See note on customizing DGOS programs for printers available from Applied Technology.

PT When listing strings (DEFM pseudo) the object field only lists the first byte of the string. This is done to conserve paper when printing.

If however you wish the object field for the string to be listed in full, use the PT switch.

The precise formats of the A instruction are shown in the command index.

PSEUDO MNEMONICS and ARITHMETIC OPERATORS

As stated earlier the assembler broadly complies with the format as defined in the ZILOG assembler, several variations are allowed to assist those familiar with 8080 assemblers. The PSEUDO operators supported are:-

ORG	nnnn	Set or redefine object address counter.
DEFB	n	Define byte to be value n.
DB	n	Same as DEFB.
DEFL	nnnn	Temporary equate label value, may be re defined later.
DEFM	'ssss'	Define contents of an ascii string.
DEFR	n	Set default radix value. 16=hex, 8=octal, 10=decimal (10 normal) if not defined, defaults to decimal values.
DEFS	nn	Reserve nn memory locations.
DEFW	nnnn	Define value of 16bit 'word' to be nn.
DW	nnnn	Same as DEFW.
EQU	nnnn	Permanent equate label value, cannot be re defined.
END		End of source listing. Stop assembly.

As well as the pseudo operators certain arithmetic operators are available for use in the operand field. these are:-

D	Consider value decimal regardless of default radix.
H	Consider value hex regardless of default radix.
O	Consider value octal regardless of default radix.

The 4 remaining operators may be used in conjuction with each other on a line, they have no assumed Hierarchy they are executed in strict left to right sequence.

+	Add the two values or labels.
-	Has two functions. When used between two labels or values it produces the difference value. When used on its own before a label it negates the value (2's complement).
&	Produces the Logical AND value of two labels or values.
<	This is the logical rotate left or right operator. The form <4 will shift the bits of the value or label left by 4 bits. The form <-5 will shift the value of the operand by 5 bits in a right direction.

There are two print directives which control listing to the line printer:-
*L ON Turns on listing. And *L OFF turns off listing. These commands are useful for printing part listings from the assembler.

GENERATING AND SAVING OBJECT CODE

We now come to the question of where you should arrange for your program to be assembled into memory, and how to save a copy of of the object code to your storage media. The answer will depend on the type of system you are running, and the type of program you are writing. In the case of MicroBee you will probably be writing mainly self contained "real time games" like the example we gave in "some hands on experience". In this situation there are very few problems since you have an area from 400hex to 1000h set aside for object code generation and use. You may just assemble your programs to run at this location, and use the "save" feature of the machine level monitor (accessed by the X command) to save the object file to cassette. Users of DGOS or CP/M versions of the assembler will however mainly wish their programs to run at 100hex, in which case any attempt to generate the object file directly to its final location will result in the destruction of the Editor/Assembler whilst it is attempting to generate the code. Obviously the offset feature will have to be used, and the finished code BLOCK MOVED to its operating location after assembly by the move function of their monitor before it can be saved. This raises two further questions, how to test the program before we wipe out the editor with the move?, and where to offset the object code to during the assembly?. It should be realised that one of the main advantages of an assembler is that the program may be made to run at any location in memory (there are a couple of rare exceptions to this rule, such as when RST instructions are used to directly reference into itself). In most cases you need only alter the ORG address and the program will run at that location. This feature is useful for testing programs that would normally be offset assembled. Now we must consider where the code may be test assembled to, or offset to. There are 4 possibilities here, in the DGOS system there is provision for an extra 1K of ram at location DC00 on the CPU card. It would be well worth the investment of fitting these two rams to the board as "safety" ram. On entry to the editor you may declare an area of "safe" ram by deliberately setting the "top of memory" address a bit lower than the real end of memory. Alternately you may leave a "hole" below your source file by starting the file at say 2800 with the Z instruction. The last possibility is to allow the code to go between the end of the source file, and the bottom of the symbol table, as we did in the example program. The only problem here is that although we can check where our file ends, there is no simple way of being sure where the symbol table comes down to. As a "rule of thumb" allowing 16 locations for each label in your program and subtracting this value from the top of memory address will give you some idea of where the symbol table will end at. There is a fifth possibility, although we don't recomend its use except as a last resort, you can overlay the start of your source file. Normally the source file will be from five to eight times longer than the object it produces, so that during the third pass when the object file is being generated, the source file read pointer is moving up the file at a faster rate than the file is being destroyed behind it. Several points must be kept in mind here. ORG the file at exactly the start address of the file. Do not use any DEFS statements near the start of the file since they move the object pointer rapidly forward. Be careful of multiple ORG statements as they too can cause the object pointer to get in front of the source pointer. Finaly, do not attempt to use any editor commands other than X or B after the assembly as the file has been destroyed.

Now that you have your object file into memory you may proceed to test run

it or save it to cassette for later use.
One final hint to CP/M users, since you needed to exit from CP/M to the system monitor to do the block move, once you have moved your object from where it was generated to, down to the TPA area at 100hex, you may reboot back into CP/M by jumping to the "WARM" boot address at address 0000. You should clear the screen before leaving the monitor since the CP/M VDU pointer may cause the prompt to occur anywhere on the screen, whilst this will not effect normal operation, it can be most disconcerting typing into the middle of old text rather than into a clear line.

EDIT ERROR MESSAGES

The following messages may occur whilst using the editor.

"String not found"
The Find or Change command could not locate the string requested. If you are sure that it should have been there, you may have started the search from after the line with the string in it, in which case go to the top of the file and try again, Or you may have spelled the word incorrectly or used the wrong alpha case.

"Command format error"
The arguments you have provided in the command line are incorrect. You may have used an illegal wild card, or forgotten to include a comma or colon etc. See the section on editor instructions for the command you wish to use

"No such line"
The line number you have requested does not exist in the currently open file. You have probably done a file renumber and the line number no longer exists. If you know any reasonably unique words or labels that exist on the line, try to locate it with the Find command. If not you will just have to step through the file with the print command till you locate the area you require.

"File full"
The end of the file has reached the upper limit of memory allocated by the answer to "Memory size?" when you entered the editor. If this is really the top of your available memory, you will have to buy more memory before you can continue, or delete some comments from the file to make it smaller. Since there is no way of re-defining the memory size from within the editor, reallocating extra memory (if available) is a little messy. You may either save the file to cassette, reboot the editor to get the "memory size?" message and reload the cassette, or exit back to a MONITOR level and adjust the memory size byte directly. A list of the location of the main scratch areas is provided in the appendices.

"Illegal command"
The command letter used at the start of the line was not recognised by the editor, or the argument to the command was in an incorrect format. See the chapter on editor instructions for the command you wish to use.

"Line number too large"
Line numbers greater than 65534 are not permitted.

"No text in file"
You have issued a command that cannot be used on any empty file. eg tried to use Replace or Edit to start inserting into the empty file. In this instance use Insert mode.

"No room between lines"

In INSERT or REPLACE modes if the next line to be inserted (after step size added) will not fit below the next existing line in the file the editor will abort with this message. To continue the insert you may either reduce the step size by I.,1<ret> or renumber the complete file by N100,10<ret>

"No file here"

You have given an instruction to re-open an OLD file at a location where the editor can't find a valid file. If you are sure that there should be a file here it is possible that some location has been corrupted (possibly bad ram or a glitch on the mains etc). If you feel competent to try to find the bug, Read the section on the layout of a file and using your monitor try to find and fix the error. You can always reset to the file with an O command on re-entry.

"No Secondary file"

You have attempted to use the COpy or Merge commands when the Editor does not have an "open" secondary file. You have either forgotten to declare a secondary file (use ZS command) or have done an interim assembly or rebooted into the editor, your old secondary file will probably still be intact and may be reopened with the OS command.

"No room for merge"

When doing a merge from secondary to primary file, the Editor must first move the end of the primary file upwards in memory to provide a 'hole' into which the secondary file would fit. If this error message is displayed, the 'gap' between the two files is smaller than the length of the secondary file, and it would have been damaged during the merge. You must therefor move the secondary file higher in memory. There are several ways of doing this, which one you should use depends on the particular situation you have. We suggest the following technique be used. (for example primary file at 2000, sec file at 3000) Query and record the status of the primary file. eg

Q<ret> 2000 2000 2F80 3FFF

Set up on the secondary file, normalise, and query its status. eg

O3000<ret> N100,10<ret> Q<ret> 3000 3000 33FD 3FFF

(In all cases you MUST ensure that the secondary file is 'normalised' by renumbering it before proceeding after setting to a secondary file.) Notice that the gap between the file is only 80hex bytes, and the secondary is nearly 400hex in length. However there is more than its own length above itself. In this case we can create a copy of the secondary file higher in memory. eg ZS3800<ret> CO#:*<ret> O2000<ret> OS3800<ret> We are now back in the original primary file with the secondary now at 3800 and plenty of room for the merge. In some cases the size of the secondary file (or its location) may not allow us enough room to make a copy above itself. If the length of the secondary file is more than twice the gap from the end of the primary file to the end of memory we cannot do the merge in one operation anyway, however all is not lost. Remember that what we used to consider the secondary file is now our primary file and may be saved to cassette. After saving a copy of this file, we may be able to open a new file (with the Z command) at a location where when the file is reloaded we will be able to do our merge. If not we can proceed to delete some of the end of it till it is small enough for the merge, then reload the saved copy of the old secondary file (at a suitable location), delete what we previously merged and remerge the remainder. There will be a lot of swapping between files, this is messy, but in an emergency justified.

ASSEMBLER ERROR MESSAGES

The following messages may occur whilst attempting to assemble a file.

"Bad label"
The "word" encountered in the label field (extreme left) does not satisfy the requirements of a label. It must not be more than six characters long and must start with an upper case alpha character. No spaces or question marks may be imbedded within the label, and it must be separated from the mnemonic field by a space or tab.

"Branch out of range"
You have used a "relative" instruction (eg JR or DJNZ) to branch to a location in your program that is more that 128 bytes away. Either rearange your program to bring the destination closer, or use an "absolute" branch instruction (eg JP).

"Illegal format"
Your line of source is not laid out out in accordance with, or contains characters not supported by, the standard ZILOG requirements. Refer to "Z8Ø-Assembly Language Programming Manual".

"Illegal opcode"
The "word" in the mnemonic field of this line is not recognised as a mnemonic or pseudo operator.

"Missing information"
The end of line was encountered before all information required had been read. You may have imbedded a semicolon in the line, or simply left out an argument.

"END missing"
The assembler found an end of file marker before the END statement, you have probably forgotten to insert one, or may have put it in the label field by mistake. Not a fatal error but will inhibit the generation of object code.

"Duplicate label"
This label has been previously defined, or may be one on the assemblers pre-defined list. eg use of HL or AF etc as labels will invoke this error message.

"Field OVF"
Whilst resolving arithmetic arguments in the operand field of a line, a value was produced that is greater than 65535 decimal (FFFFhex).

"Ref duplicate label"
This message will be invoked on all lines containing reference to duplicated labels.

"Symbol table OVF"
The symbol table being produced in pass 1 of the assembly has grown down to the point where, if assembly continued, the source code would be damaged. This is a fatal error and assembly is immediately aborted.

"Label not known"
You have attempted to reference a label which has not been defined in the label field. Usually invoked by spelling mistakes, or forgetting to assign scratch locations.

"Expression error"
The operand (address or value) field could not be resolved. It may contain a value or character that is not supported in the current radix default (eg alpha character in decimal expression) or applying a 16 bit mask to an 8 bit value.

SOME HELPFUL HINTS

Probably the most helpful hint that can ever be given is that regularly creating backup copies is the best means of preventing major disasters. It has been stated that while ever mans fingers are pointed towards a keyboard they will occasionally get in front of his brain. Imagine wishing to print your latest hours of typing with an T#:* and accidentally pressing D#:* thereby deleting the entire file. It has happened!!! It may be the last backup was made a half an hour ago, but it it always easier to redo the last half hours work than to start from the beginning.

Get into the habit of always starting your source files with a comment line which has the file name, and or description, **AND THE DATE.** It often happens that when you wish to reload from an old file there are several backup copies of it in existence. The date, and if desired the time, of the save will help to identify the most recent copy.

Always do your trial assemblies with the WE and NO switches specified. This will ensure that no crashes occur, and that you must attend to each error as it is reported.

Sometimes an error will be reported on a line, and yet you cant see anything wrong. It is possible that a non printable character may have been inserted into the line by either an editing error, or a power glitch. The best solution here is to delete the line and type it in again. If the error is still there, re read the manuals for the type of line you are inserting. The error message will usually help to pin it down.

If you wish to delete a line from the file, use the D command, don't try to edit it back to nothing with the backspace key, this can cause some unpredictable crashes under certain circumstances.

The two most common problems with assembled programs that don't appear to work, is forgetting to append a H to hex values, particularly when EQUating monitor calls, and not keeping the number of PUSH's and POP's balanced in sub routines. If your programs seem to crash in a great heap, try looking at these two problem areas first.

The line numbers inserted onto each line by the editor are not recognised by the assembler as labels, therefore you cannot use them as symbolic references for CALLS or JUMPS etc as you would in BASIC.

If whilst trying to list a file to the VDU the P command refuses to go past a certain point in the file, or appears to be looping back on itself, try renumbering the file, this will usually fix the problem. What has happened is that a faulty memory location (or a glitch) has caused a line number in the file to appear to be lower than the one before it. and the editor has become confused.

The "end of file marker" is two bytes containing FF FF, this represents line number 65534 and all other lines will be placed before it. If for any reason one of these two bytes gets damaged, the file will appear to continue on into whatever rubbish happens to be in memory. The fix here is to step forward gradually through the file till you are sitting on the line before the crash, then use the Q command to locate your position in memory, and

then use the system monitor to replace the two FF's at the end of the file. See appendix B for the exact layout of a file to get a better understanding of the problem. Remember that most crashes can be recovered from if you use a little thought before proceeding. Rushing into the "fix" will often only compound the problem.

GLOSSARY OF TERMS

ADDRESS Describes an actual memory location in the computer. With assemblers it is normal to refer to addresses in HEXADECIMAL notation. If the address starts with a letter it is correct procedure to prefix the address with a zero to avoid confusion with a word or label. eg ØBAD is an address.
APPEND Add to the end of. eg append a comment to a line.
ARGUMENT The value, address, or name that we wish the assembler to assign to the particular field. The argument may be a complex statement comprising arithmetic or logical operators.
ASSEMBLER See the chapter "what is an assembler".
BASIC An "english word" oriented interpreter, used to allow people not experienced in advanced programming techniques to create and run their own computer programs.
BLOCK MOVE A command that allows you to physically move the location of a file or collection of bytes.
BUFFER An area of memory set aside for storage and processing of commands, editing of lines, resolving argument fields etc.
BYTE An 8 bit hexadecimal number (ØØ to FF) which represents the 256 different values that can be stored in one location of computer memory.
COMPILER Any program which can produce a machine or object code output from a mnemonic source file.
COPY A command which allows you to duplicate a line, or lines, into another part of memory.
CP/M A disk based operating system for 8080 and Z80 processors. (copyright by Digital Research U.S.A.).
CURSOR A pointer to your current location in the line or file, usually shown as a flashing square on the display.
CONTROL CODE These are special key codes typed on the keyboard to instruct the program to perform a certain task. Sometimes a special key is provided, other times you must simultaneously press the CONTROL key and a letter key. the TAB key is the same as control I. abbreviated as CntrlI or ^I.
DEFAULT The condition or value that the program assumes in the absence of a specific value being given by the user.
DELETE The act of removing a line or group of lines from the file.
DELIMITER A character (usually <ret> or /) used to signify the end of the line, or argument within the line.
DGOS A machine language operating system sold by Applied Technology for use with their S1ØØ processor series. (DGOS is copyright by Mr D. Griffiths).
EDIT The method of altering, or correcting a line in the file.
EDITOR See the chapter "what is an editor".
EQUATE To assign a value to. To define the meaning of.
FIELD Refers to a sub section of a line. eg label field, mnemonic field, operand field, comment field.
FILE Any collection of words, letters, characters, numbers or data stored on cassette, disk, or in the computer's memory.

FREEFORM Means that the program is not affected by the precise layout of your entry within certain constraints, eg you may use single or multiple spaces instead of tabs etc.
GLOBAL All inclusive, not just limited to the area you are working in.
HEX or HEXADECIMAL A system of counting with a base of 16 rather than the more familiar base of 10, comprises the digits 0 to 9 followed by A to F.
INSERT The act of providing a new line, or lines into the file.
LABEL A "word" or group of characters, starting with a letter, which defines a particular location or value.
LINE The collection of words starting with the LINE number and ended by pressing the the <ret> key that comprise an instruction, or command, for the assembler to process.
MACHINE CODE The collection of hexidecimal numbers read directly by the processor that comprise a program. Also referred to as OBJECT CODE.
MERGE The act of bringing back into the file a line, or group of lines, from elsewhere in memory. The converse of COPY.
MICROBEE A small, self contained, Z80 based microprocessor, sold by Applied Technology Pty Ltd.
MNEMONIC A "word" or symbol, often heavily abbreviated which refers to a specific task you wish the computer to perform.
MONITOR A program (usually in EPROM) used to perform the essential tasks of loading tapes, printing to the VDU, reading the keyboard, etc.
NULL LINE An empty line, used to visually break up the program into modules. This is created by pressing <ret> only as a line entry. Null lines are ignored by the assembler.
OBJECT CODE The collection of hex numbers that comprise a computer program, the output from the assembler is object code, also referred to as MACHINE CODE.
OFFSET A constant value added to the address when the assembler is outputing the object code, this causes it to be stored in a different memory location from the one where it would normally be run.
OPERAND The value field. It is in this field that the value to be assigned, or the address to be used is determined.
ORG The origin address specified in your source file that directs the assembler as to where the program is required to operate in memory.
PATCH An alteration to the program that is done outside the main body of the program. Usually placed at a location where it is convenient for the user to be able to modify, or customise the program for his own needs.
PERIPHERALS Additional devices connected to the computer to perform specific tasks. PRINTERS, CASSETTE recorders, etc are examples of peripheral devices.
PROGRAM A group of commands, or instructions, that direct a processor to perform a specific task.
PRINTER The computer "Hard copy" device, an electronic typewriter of some form, connected to the computer.
PSEUDO Literally 'false'. Pseudo mnemonics, although not really a defined mnemonic, are used in the mnemonic field to specify certain tasks to be performed.
REPLACE This command deletes one line from the file, and allows you to insert a new line, or lines in its place.
RENUMBER The automatic process of re-adjusting all the line numbers so that they are in ascending sequence, with equally spaced steps.
SCREEN This is an alternate word used to describe the VDU or T.V. set.
SOURCE FILE The list of instructions, created with the editor, and read by the assembler, to create your machine language program.
SOURCE LISTING The printed listing from the assembler that shows the source

file with the addresses and object code produced for each line.
STATUS A display of the START, CURRENT POINTER, and END addresses of the file, plus the currently set upper limit of memory.
STRING A group of characters, similar to a sentence in normal speech.
SWITCHES Two letter groups, used to direct the assembler to perform particular tasks during assembly, if not specified, each switch is considered to be in a OFF state.
SUB EDIT Whilst in EDIT mode, an alternate set of command letters are used to direct the processor, these are called SUB EDIT instructions.
SYSTEM A collective term refering to the group of components which make up your "computer".
VDU The computer display device, usually a modified TV receiver.
VIEW The ability to look around the area you are currently working in without altering the current pointers.
WILD CARDS Characters used to perform a task within certain broadly specified restraints, eg P.:* means display from current location to end of the file, or G* means load in any program from tape regardless of name or type.
ZILOG The American company who developed and produced the Z80 processor system, and laid down the preferred MNEMONICS to be used to refer to its many operation codes.
Z80 A CPU chip which is an advanced, upwards compatable, version of the 8080 series of processors. As well as its own instructions, a Z80 can also execute all of the 8080 instructions.
8080 A CPU chip developed by INTEL U.S.A. The predecessor of the Z80 processor that you are currently running.

COMMAND INDEX

Editor.

B	Exit to 'monitor'	Q	Query file status
C	Change /string1/string2/	R	Replace lines
CO	Copy to secondary file	S	Save file called "NAME"
D	Delete lines from primary file	T	Type, no line numbers
E	Enter edit sub mode	V	View lines around current
F	Find /string/	X	Exit from editor to address
G	Get from tape "NAME" or *	Z	Create new primary file
I	Enter insert mode	ZS	Create new secondary file
L	List to printer	^	Step back one line
M	Merge from secondary file	`	Freeze VDU during scroll
N	Renumber primary file	B/S	Destructive backspace
O	Open old primary file	L/F	Step forward one line
OS	Open old secondary file	C/R	End of line character
P	Print file to VDU	CtlC	Exit insert mode

Sub edit

A	Ignore changes, restart edit	L	List full line, restart edit
nC	Change next n characters	Q	Quit do not alter original line
nD	Delete next n characters	nSx	Move to n'th occurrence of x
E	End edit include changes	X	Insert from end of line
H	Delete rest of line, & insert	SPACE	Non destructive move right
I	Insert string into line	B/S	Destructive move left
nKx	Kill all chars to the n'th x	C/R	End edit return to editor

Assembler

A <ret> Normal assembly with object produced (space between A and <ret>)
Annnn<ret> Assemble with object offset in memory by nnnn
A/S1/S2/S3/S?<ret> Assemble with switch control
Annnn /S1/S2/S?<ret> Assemble with offset under switch control

` Freeze display during listing, any other key continues listing.

Switches

WE	Wait if error found	NS	No symbol table displayed
NO	No object code produced	LP	Produce full listing to printer
NL	No source listing	PT	Print strings in full

After error encountered during assembly

Control C	Return to editor with error line as current editor line
C	Clear down error switch (no wait if extra errors)
Any other key	Continue assembly with error switch still on

MICROBEE MONITOR

Since the MICROBEE in its standard form is essentilaay a "BASIC" only microcomputer with the ability to run pre-recorded machine code programs, it was decided to provide the functions of a monitor program to enable the user to create and run his own programs at machine code level.

To use the monitor press the 'M' key and 'RESET' at the same time and then, while still holding down the 'M' key, release the 'RESET' key and the MICROBEE should respond with:

```
          Microbee Monitor V1.Ø
>
```

You can enter the commands as follows:

B — Return to BASIC

E nnnn — Examine hex memory location nnnn. To move the cursor use ^S to step to the right, ^A to move to the left, ^W to move up the screen, ^Z to move down. To **MODIFY** the location currently indicated at the cursor position, type **M** followed by the byte (in hexadecimal) you want to insert into memory.

G nnnn — Go tp address nnnn and run the program

M nnnn oooo pppp — Move a block of data pppp long from address nnnn to oooo.

R — Load a tape

S nnnn oooo vv ww xx yy — Search through memory from nnnn to oooo for specified bytes vvwwxxzz.

W "NAME" M nnnn oooo pppp — Save a machine code program starting at nnnn, ending at oooo with auto execute at pppp. If address pppp is not included the program will default to the first address nnnn as the auto start address.

NOTE: If you enter a character that is not 'understood' by the monitor a flashing '?' will appear. Use 'BACKSPACE' or 'ESC' key to cancel the error and return to the monitor command mode.

APPENDIX A "test source list"

This is the test program you are required to enter for the chapter "some hands on experience". Note that you enter exactly what is shown here with the exception of the numbers shown in the right hand column; these are shown for your convenience and should match the number automatically inserted by the editor at the start of each line. If you make a mistake whilst typing a line you may use the backspace key to fix it, If you do not notice it till after you finish the line, wait till we have shown you how to EDIT lines.

```
;Test program for "skywriter"                                    00100
VDU     EQU     0F000H                                          00110
SPACE   EQU     20H                                             00120
        ORG     3000H                                           00130
START   LD      HL,VDU                                          00140
        LD      A,'A'                                           00150
        LD      (CHAR),A                                        00160
CLEAR   LD      (HL),SPACE      ;Clear top of VDU               00170
        INC     HL                                              00180
        LD      A,L                                             00190
        OR      A                                               00200
        JR      NZ,CLEAR                                        00210
        LD      HL,VDU                                          00220
FLY     LD      DE,0            ;Speed value                    00230
WAIT    DEC     DE              ;Slow down display              00240
        LD      A,D                                             00250
        OR      E                                               00260
        JR      NZ,WAIT                                         00270
        LD      A,(CHAR)                                        00280
        LD      (HL),A          ;Put char to VDU                00290
        INC     A                                               00300
        CP      'Z'+1                                           00310
        JR      C,STORE                                         00320
        LD      A,'A'                                           00330
STORE   LD      (CHAR),A                                        00340
        INC     L                                               00350
        PUSH    HL                                              00360
        CALL    PLANE                                           00370
        POP     HL                                              00380
        JR      FLY                                             00390
PLANE   LD      DE,STRNG        ;There is an error here         00400
PRINT   LD      A,(DE)                                          00410
        OR      A                                               00420
        RET     Z                                               00430
        LD      (HL),A                                          00440
        INC     L                                               00450
        INC     DE                                              00460
        JR      PRINT                                           00470
STRING  DB      '>'                                             00480
        DB      0A              ;Another error                  00490
        DB      9                                               00500
        DB      0                                               00510
CHAR    DB      'A'                                             00520
        END                                                     00530
```

APPENDIX B Layout of a file.

For those who wish to attempt to recover a "crashed" file, or convert a differently formatted file for use with this editor, the precise layout is defined here.

There is no start of file character, nor is there any end of line character stored on the line, the lines are stored sequentialy, in ascending order of line numbers. Each line consists of a 16 bit value representing the line number, normal 8080/Z80 address format is used, that is the low byte first. eg line number 100 will appear as 64 00 (100 decimal is 0064hex). Following the line number is a single hex byte representing the number of characters (not counting the line number or itself) that are in the line. The actual characters (in hex) of the line follow the length byte. After the last line of the file is the "end of file marker" which consists of 2 bytes, both FF, the end of file marker serves two purposes, FF FF represents line number 65535, and ensures that all other lines will be sorted below itself. It also represents the end of the file. Note that the end of file pointer displayed by the Q command shows the location of the first of these two bytes. And any attempt to save the file from outside the environment of the editor must include both FF's. To satisfy the editor that a valid file exists, It must be capable of stepping through the file by means of the length bytes, until it finds the end of file marker below the end of memory address. Whilst stepping through it must not find a length byte longer than 7Fhex, nor must there be any characters in the line that have the sign bit set (reverse video). If any of these criteria are not met, the message "No file here" will be displayed.

The following is a sample file, at 2000hex, and the memory image produced.

After printing the file, the status displayed by the Q command was:-
Q<ret> 2000 2026 202D 3FFF

```
00100;TEST LINE
00110       ORG    100H
00120       JP     0D000H
00130       END
```

```
2000    64 00 0A   3B 54 45 53 54 20 4C 49 4E 45
200D    6E 00 09   09 4F 52 47 09 31 30 30 48
2019    78 00 0A   09 4A 50 09 30 44 30 30 30 48
2026    82 00 04   09 45 4E 44
202D    FF FF
```

APPENDIX C Editor scratch layout.

For DGOS and CP/M users who may wish to customise their "personality" modules, we present a list of the major scratch locations that they may wish to reference into. MicroBee owners should consider the list starting at 0200 rather than at 1E00. The list is not comprehensive, but mearly covers those you are most likely to wish access to. When shown as xxxx/y it means that a 16 bit value is used.

1E00/1	Pointer to start of currently open primary file.
1E28	Secondary file open flag.
1E29/A	Start of secondary file pointer.
1E2B/C	End of secondary file pointer.
1E2D/E	Current line pointer.
1E2F/0	End of memory pointer. Set up on entry to editor.
1E31/2	End of file pointer (points to first FF)
1E33	Step size between line numbers.
1E43/C3	128 character general purpose buffer.
1EC4/5	Pointer to current character being accessed in buffer.
1EC6	Size of line currently in buffer.
1ED4/5	Error count. Keeps count of number of errors during assembly.
1ED6	I/O suppress flag. Non zero suppresses output to both VDU and line printer. Used to suppress listings under NL switch etc.
1ED8	Printer flag. Non zero causes printer to be used instead of VDU.
1EDF/0	Pointer to bottom of Symbol table. Only valid after completion of an assembly.
1EE1	Suppress line numbers on printout if non zero.
1EE2	File initialised flag. Editor will ask for "memory size", and create a null file at the default file address if this location is not set to 55hex on entry to the Editor.
1EE9	Current value default radix.
1EEA	Reserved for line number of printouts.
1EEB	Reserved for page number of printouts.
1EF0/5	Assembler switch storage scratch.
1EF7/06	Cassette name compare buffer.
1F07/16	Cassette header and address buffer.
1F17/97	Return address stack.
1F98---	Spare locations to 1FFF.